ML-based premise selection for Lean

Ramon Fernández Mir

School of Informatics
University of Edinburgh

ramon.fernandezmir@ed.ac.uk

10th March 2023

Problem description

```
example : 2^(n + 1) * m = 2 * 2^n * m := by {
  -- What now ?
}
```

We just need to use the theorem that says that $2^{n+1} = 2 \cdot 2^n$ (`pow_succ`).

Or, even better, have the system prove it automatically.

Issues:

- `mathlib` has over 100k theorems.
- There are ways to search but they are very strict.

Solution

Turn this problem into a machine learning task where:

- **Input**: the theorem statement (featurized).
- **Output**: list of premises that appear in the proof.

Design principles:

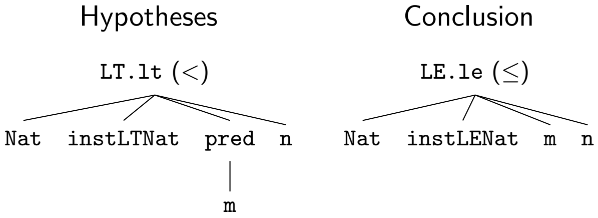
1. Tight integration with the proof assistant.
2. Easy to use and install.
3. Lightweight and fast.

Data extraction, training and prediction all happen in Lean.

Features

```
theorem le_of_pred_lt {m n : ℕ} : pred m < n → m ≤ n := ...
```

These are well-defined expressions, so we consider their syntax tree:



- Names: T:LE.le T:instLENat T:Nat H:Nat H:LT.lt H:instLTNat ...
- Bigrams: T:LE.le/instLENat T:LE.le/Nat H:LT.lt/Nat ...
- Trigrams: T:LE.le/Nat/instLENat H:LT.lt/Nat/instLTNat ...

Relevant premises

The proof is also an expression so, in principle, we could just traverse it and keep track of all the premises found.

However, this results in a large number of simple facts and autogenerated lemmas...

We apply three filters[1]:

- All (42k): remove premises automatically generated by Lean.
- Math (40k): remove premises from the core library, e.g. `rfl`.
- Source (21k): only keep lemmas explicitly written in the proof.

```
match m with
  | 0 => le_of_lt
  | m + 1 => id
```

Relevant premises

The proof is also an expression so, in principle, we could just traverse it and keep track of all the premises found.

However, this results in a large number of simple facts and autogenerated lemmas...

We apply three filters[1]:

- All (42k): remove premises automatically generated by Lean.
- Math (40k): remove premises from the core library, e.g. `rfl`.
- Source (21k): only keep lemmas explicitly written in the proof.

```
match m with
  | 0 => le_of_lt
  | m + 1 => id
```

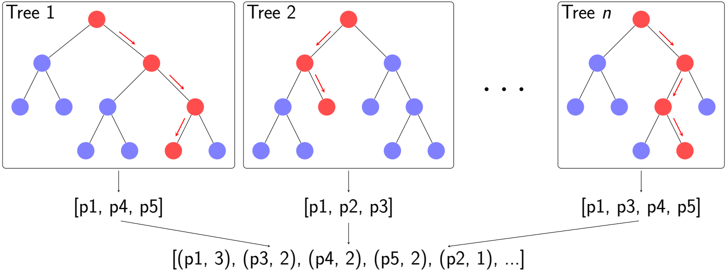
[1] In brackets: number of theorems with non-empty premise lists after filtering

Random forest

Key idea: many (uncorrelated) decision trees + voting.

Our decision trees:

- Leaves hold a list of premises and a list of examples.
- Nodes consist of a simple rule checking if a feature appears.
- The output is a ranking of premises.



Random forest

A key difference with the usual approach is that we train it in an *online* fashion, i.e. we update the model one example at a time. It makes it easy to update the model as new theorems are proved.

The steps to add an example e to a tree are:

1. Follow the binary rules down to a leaf L.
2. Let $L = L \cup \{e\}$. If $split(L)$, continue, else stop.
3. Select N features by successively taking random pairs of examples in L and picking a feature in their difference set.
4. The new rule f is the feature maximizing "information gain".
5. Split L based on f into L_1 and L_2 and let $L = (f, L_1, L_2)$.

Evaluation and results

Split training and test sets based on `mathlib` modules:

- Test (592): Modules that are not dependencies.
- Training (2436): The rest of the modules.

Assume a theorem T depends on a set P of n premises. We measure the quality of a ranking R as follows:

$$Cover(T) := \frac{|P \cap \{R[0], \dots, R[n-1]\}|}{n}$$

We also consider taking $n + 10$ premises from R instead of n.

Evaluation and results

Average cover for our model with different filters and features:

	`n`	`n+b`	`n+b+t`
All	0.56 (0.67)	**0.57** (0.67)	0.47 (0.58)
Source	0.28 (0.36)	**0.29** (0.36)	0.28 (0.36)
Math	0.25 (0.32)	**0.26** (0.33)	0.16 (0.24)

Observations:

- More strict filters make the learning task harder.
  - Fewer data points.
  - It is "easy" to predict very common premises.
- Trigrams caused over-fitting.

Demo

Project summary

Co-authors:

- Bartosz Piotrowski (University of Warsaw)
- Edward Ayers (Carnegie Mellon University)

The code is publicly available at:

`https://github.com/BartoszPiotrowski/lean-premise-selection`

Future work:

1. Better features exploiting the structure of expressions.
2. Use our ML advisor to guide automated reasoning tools.
3. Can a more sophisticated model get better results?

Related projects

Tactician (2021)

- Tactic selection for Coq.
- Tutorial: `https://coq-tactician.github.io/`.

Thor (2022)

- Premise selection using a language model.
- Works with automated theorem provers (hammers).

I also recommend Jason Rute's recent talk "Deep learning in interactive theorem proving" for more projects in this direction: `https://www.youtube.com/watch?v=P5ew0BrRm_I`

Thank you